
ELSEVIER

Parallel Computing 26 (2000) 1231–1252

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Real-time sonar beamforming on high-performance distributed computers

Alan D. George *, Jeff Markwell, Ryan Fogarty

Department of Electrical and Computer Engineering, High-performance Computing and Simulation (HCS) Research Laboratory, University of Florida, P.O. Box 116200, Gainesville, FL 32611-6200, USA

Received 31 August 1998; received in revised form 3 November 1999; accepted 5 January 2000

Abstract

Rapid advancements in acoustical beamforming techniques for array signal processing are producing algorithms with increased levels of computational complexity. Concomitantly, autonomous arrays capable of performing most or all of the processing in situ have become a focus for mission-critical applications. To address these changes, future sonar systems will take advantage of parallel in-array processing by coupling transducer nodes with low-power processing devices to achieve higher performance and fault tolerance at lower cost. This paper explores parallel algorithms for conventional beamforming (CBF) designed for an in-array processing system. The parallel algorithms presented offer scaled speedup and provide the basis for adaptations in advanced beamforming algorithms.

Keywords: Parallel beamforming; Parallel sonar; Real-time sonar; Distributed processing; In-array processing

1. Introduction

Quiet submarine threats and high clutter in the littoral undersea environment demand that higher-gain acoustic sensors be deployed for undersea surveillance. The effect of this trend is high-element-count sonar arrays with increasing data rates and associated signal processing. The US Navy is developing low-cost, disposable, battery-powered, rapidly deployable sonar arrays. These autonomous, passive, sonar

* Corresponding author.
E-mail address: george@hcs.ufl.edu (A.D. George).

0167-8191/00/$ - see front matter
PII: S0167-8191(00)00037-5

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2000		**00-00-2000 to 00-00-2000**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Real-time sonar beamforming in high-performance distributed computers	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Florida,Department of Electrical and Computer Engineering,High-performance Computing and Simulation (HCS) Research Laboratory,Gainesville,FL,32611**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**22**	

array technologies face difficulties, such as low fault tolerance due to single points of failure and computational complexity not readily supported in real-time by conventional means that may be overcome with the use of parallel and distributed computing (PDC) technology. These challenges in achieving critical levels of performance and reliability are particularly difficult to overcome for systems that employ high-fidelity beamforming algorithms such as adaptive and matched-field processing and must process data from a large number of sensor nodes.

The next generation of passive array systems will support powerful signal processing algorithms, provide fault-tolerant mechanisms to overcome node and communication link failures, and operate at very low power levels so that they may function on battery power for mission times measured in weeks or months. These arrays will be capable of uplinking real-time beamformed data with better resolution than conventional systems while lowering cost by exploiting commercial off-the-shelf (COTS) microprocessors and subsystems. This paper presents new parallel algorithms for conventional beamforming (CBF) optimized for future in-array processing sonar systems. These embedded technologies have advantages in both performance and reliability as the computational capacity scales with array size and fault tolerance is increased by the elimination of single points of failure.

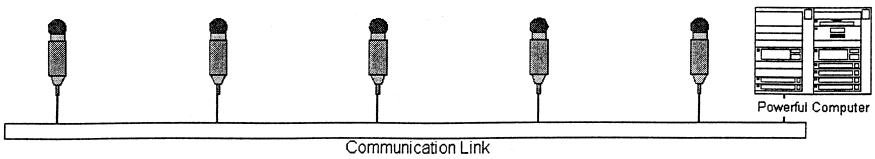
Many optimizations exist for CBF that improve the algorithm complexity. Mucci presents seven categories of CBF algorithms, each having different spectral characteristics and hardware requirements. These categories include delay-and-sum, partial-sum, interpolation, interpolation with complex sampling, shifted-sideband, discrete Fourier transform (DFT), and phase-shift beamformers [15]. Although other optimizations exist, such as Houston's fast beamforming (FBF) algorithm [8], these seven can serve as a basis. Each algorithm is related to the fundamental delay-and-sum CBF algorithm. The first five algorithms work in the time domain while the latter two (i.e., DFT and phase-shift beamforming) work in the frequency domain. The delay-and-sum algorithm has the unfortunate limitation that the spatial resolution is dependent on sampling frequency, resulting in a very large data space when large numbers of steering directions are desired. Each of the other algorithms aforementioned is optimized by minimizing this characteristic to varying degrees.

The DFT beamforming algorithm was chosen as the algorithm for the in-array parallel processing system. Advantages of the DFT beamformer include its ability to be updated for use with adaptive algorithms such as the minimum variance distortionless response (MVDR) beamformer [11]. Also, inverse transforming is not required since the frequency information is often advantageous for signal detection, localization, and classification in post-processing objectives [15]. DFT beamforming algorithms have the ability to support any number of steering directions at arbitrary angles, and take advantage of the efficiency inherent in the fast Fourier transform (FFT) algorithm. McMahon discusses some of the disadvantages of the DFT beamformer including the inability of the Fourier domain to track certain classes of pulses, which is a general limitation of high-resolution Fourier analysis [12]. Also, even though the memory space required for the DFT beamformer is smaller than that for the delay-and-sum beamformer, Mucci points out that the DFT has a large data space in comparison to memory-efficient methods such as partial-sum, the

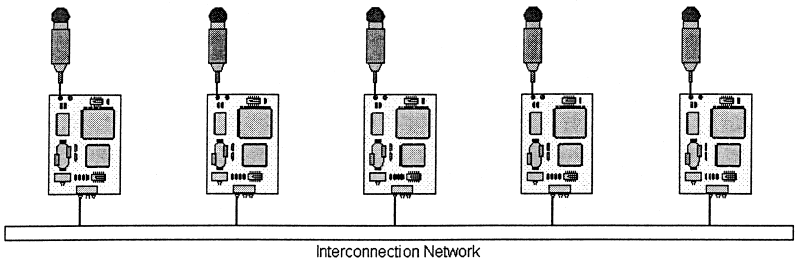
interpolation techniques, and the phase-shift algorithm. Despite these limitations, DFT beamforming algorithms are attractive for parallel sonar arrays.

Conventional arrays may be described as a string of "dumb nodes" (i.e., nodes without processing power) with a large front-end processor, shown in Fig. 1(a). Alternately, these dumb nodes may be outfitted with intelligent COTS microprocessors, shown in Fig. 1(b). Emerging technologies for sonar signal processing arrays will exploit such intelligent distributed-memory multicomputer systems. These systems are typically programmed in a multiple instruction multiple data (MIMD) [4] fashion using a message-passing paradigm. Coarse-grained parallel decompositions are usually the preferred approach on distributed-memory multicomputers; however, medium-grained algorithms are also feasible with the advent of fast interconnection networks with lightweight communications. The coarse-grained and medium-grained parallel CBF algorithms introduced in this paper can be used as the foundation for more sophisticated beamformers that one day will be targeted for intelligent array systems. These advanced techniques, in increasing order of complexity, include split-aperture beamforming, adaptive beamforming, matched-field tracking, and matched-field processing. The conventional beamformer exploited and parallelized in this paper will remain as a fundamental portion of such future algorithms, and thus the parallel algorithms presented can serve as a basis for all future work in this field.

Other research initiatives in the parallelization of beamforming algorithms include decompositions of conventional techniques over tightly coupled shared-memory multiprocessors. Although the amount of digital signal processing (DSP) research on array processing is abundant, little has been accomplished in the area of MIMD-style decompositions. Several projects from the naval undersea warfare center



a. Conventional passive sonar array



b. Distributed parallel sonar array

Fig. 1. Conventional passive array vs distributed parallel sonar array.

(NUWC) have developed real-time sonar systems by exploiting massive parallelism. Salinas and Bernecky [17] mapped the delay-and-sum beamformer to a MasPar, a single instruction multiple data (SIMD) architecture. Dwyer used the same MasPar machine to develop an active sonar system [3]. Lastly, Zvara built an active sonar processing system on connection machine's SIMD architecture, the CM 200 [22]. A handful of other papers discuss a variety of adaptive algorithms on parallel systems including systolic arrays, SIMD style machines, and COTS-based DSP multicomputers [1,19]. However, little of the previous work has focused on issues involved in either distributed or in-array processing.

This paper introduces three parallel DFT beamforming algorithms for distributed in-array processing: iteration decomposition, angle decomposition, and pipelined angle decomposition. In Section 2, the reader is presented with basic beamform theory and the sequential DFT beamformer. In Section 3, the parallel iteration-decomposition beamformer, which is based on a pipelined approach, is introduced. In Section 4, angle decomposition, a data-parallel approach, is discussed. The third parallel beamforming algorithm, pipelined angle decomposition, is presented in Section 5 and is shown to be a hybrid of the first two parallel algorithms. The sensitivity of each algorithm to array parameters is shown in Section 6 using a CAD-based rapid virtual prototyping environment. Finally, in Section 7, a brief set of conclusions and directions for future research is enumerated.

2. Conventional beamforming

Conventional delay-and-sum beamforming may be performed in either the frequency or time domain. In either domain, the algorithm is essentially the same: signals sampled across an array are phased (i.e., delayed) to steer the array in a certain direction, after which the phased signals are added together. To phase the incoming signals, some geometry is needed to transform the steered "look" direction to the correct amount of delay at each node. Fig. 2 shows an incoming plane wave and the geometry needed to derive the delay.

This delay is directly proportional to Δx as shown in the figure. Adjacent nodes would receive the wave front at a difference in time of

$$\delta = \frac{\Delta x}{c} = \frac{d \sin(\theta)}{c}, \tag{2.1}$$

where d is the distance between adjacent nodes, θ the steering angle from the array's perpendicular (i.e., from broadside), and c is the speed of sound in water (estimated as 1500 m/s). Each node, indicated by its node number m, would receive a signal from angle θ at a relative delay of

$$(m-1)\delta, \tag{2.2}$$

with respect to node zero. In the frequency domain, a vector $\underline{s}$ may be built which, when multiplied by the respective incoming signals, will properly phase the system. This vector is defined as

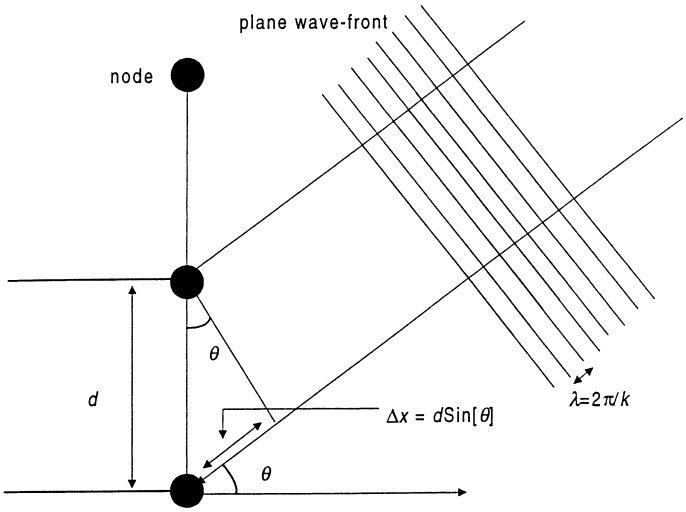


Fig. 2. Wave fronts hitting array.

$$\underline{s} = [1, \quad e^{-jkd\sin(\theta)}, \quad e^{-j2kd\sin(\theta)}, \ldots, \quad e^{-j(M-1)kd\sin(\theta)}], \tag{2.3}$$

where

$$k = \frac{2\pi}{\lambda} = \frac{\omega}{c} \tag{2.4}$$

is equal to the wave number and M is the number of nodes. The final beamform equation is a summation, $\underline{x}(t)$, of the phased signals multiplied by a windowing weight matrix $\underline{w}^t$

$$y(t) = \underline{w}^t \underline{x}(t) \quad \text{(time domain)} \tag{2.5}$$

or

$$y(\omega) = \underline{w}^{\omega} \underline{x}(\omega) \quad \text{(frequency domain)}. \tag{2.6}$$

Readers interested in a more complete discussion of beamforming algorithms are referred to [2,7,10].

The baseline DFT beamforming algorithm, adapted from [16], consists of five operations: a window multiplication, DFT (via the radix-2 FFT adapted from [14]), steering factor (i.e., phasing) multiplication, beamforming summation, and last, computation of each angle's power and inverse Fourier transform. Though not necessary for some types of post-processing, the inverse transform is included in order to output the beamformed time series. The steps are shown in Fig. 3, where each of the operations is annotated with the dimension map of the data object produced by the block. The size of the matrices and vectors involved are defined in terms of the following parameters: *number of nodes* (M), *input sample size* (L), *number of frequency bins* (N), and *number of steering directions* (S). The nodes are the sensors distributed across the length of the sonar array, and the input samples are the

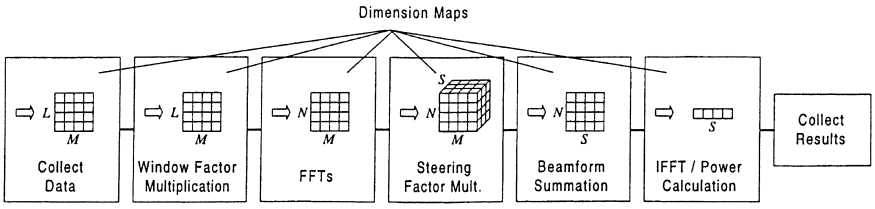


Fig. 3. Flowchart for sequential DFT beamformer.

time-domain data values collected from the sensors. The frequency bins employed in a given beamformer depend upon the nature and number of input signals of interest and the intervals in frequency they occupy, whereas the steering directions are the angles used in the beamforming process.

The *window factor multiplication* stage scales each node's input by some factor. For CBF, a windowing function such as Hanning, Blackman, or Dolph–Chebyshev is typically applied across the hydrophone array such that nodes toward the ends of the array contribute less in the *beamform summation* stage. The computations involved include $L \times M$ floating-point operations. This method improves the signal-to-interference ratio and is analogous to windowing on Fourier transformations [18]. Adaptive techniques adjust these scalar weights to optimize the beamform solution by minimizing output power in a constrained manner [5,21].

The *FFTs* stage takes the scaled temporal data stream and converts it into complex-valued spectra. The computations involved include M distinct FFT operations, each comprised of $O(L \times \log L)$ floating-point operations. Since the amount of data entering this stage equals the amount of data exiting, it is reasonable to assume that this operation should be performed within each node to preserve the data parallelism inherent in the system (i.e., each node transforms its own sample sets).

The *steering factor multiplication* stage is a major computational bottleneck of DFT beamforming and is not required in time-domain algorithms. The computations involved include $M \times N \times S$ complex multiplications. It is interesting to note that this steering factor multiplication stage is exactly the same operation as correlation by plane-wave replica vectors and thus lead to matched-field techniques. The steering factor matrix may be recomputed with each iteration of the beamformer, or it may simply be stored and retrieved from memory. The former method will result in a memory-efficient model while the latter will result in a computation-efficient model.

The beamform summation stage represents another major computational bottleneck. However, while this stage shares the sample computational complexity as the previous stage, it is comprised of complex additions that are typically less work than the comparable number of complex multiplies required in the steering factor multiplication stage. The beamform summation stage is the actual spatial filter, and thus the reduced output data is ideally filtered spectra for each angle of interest. This operation is the same for any beamforming method in time or frequency, using conventional or adaptive techniques.

Finally, the *IFFT/power calculation* stage transforms the data back into the time-domain and calculates the power of the beamformed spectra for each steering direction. The computations involved include S distinct inverse FFT (IFFT) operations, each comprised of $O(L \times \log L)$ floating-point operations, along with $O(L \times S)$ floating-point operations to compute the power. This work is often left for post-processing stages or completely ignored because spectral information is many times more useful than temporal data when human operators (or pattern recognition post-processors) analyze the data. Nonetheless, for completeness and to better support in-array processing, both the IFFT operation and the power calculation are included in all our algorithms.

To further analyze the performance characterisitics of this beamformer and its stages, the sequential baseline algorithm was coded and executed on an UltraSPARC-1 workstation running at 170 MHz with 128 MB of memory. Fig. 4 shows the algorithm's performance when computing 91 and 181 steering directions for different numbers of input sensors (i.e., problem sizes). The top of each stacked bar represents the total time to complete, and each bar is partitioned by the five stages discussed above (where the window factor multiplication is combined with the FFT stage and the inverse FFT is combined with the beamform summation stage), with *other* representing the overhead of the timing functions employed for the measurements. As predicted, the steering factor multiplication stage represents the largest bottleneck followed by the beamform summation stage. The remaining stages represent a small fraction of the total computation time. This algorithm was optimized for memory efficiency by recomputing the steering matrix factors at each iteration; therefore, the execution time of the steering multiplication could be significantly decreased if desired, but with the penalty of a larger memory requirement. For this sequential baseline, we chose to ignore the communication time required to gather samples from a passive sonar array with dumb nodes. Since communication in the sequential system is primarily for the collection of input samples, and sonar systems typically function with a relatively low rate of input sampling (e.g., in units of kHz), an objective of complete overlap of communication by computation to hide communication latency is easily realized on traditional systems.

In order to maximize the efficiency of the parallel DFT beamforming algorithms, the steering factor multiplication and beamform summation stages must be

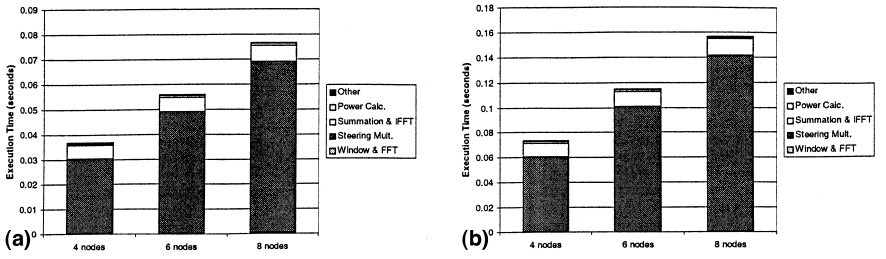


Fig. 4. Average execution times for the DFT beamformer (averaged over 1000 iterations). (a) 91 steering direction, (b) 181 steering direction.

parallelized so as to overlap or partition these computations while simultaneously minimizing communication requirements. Each of the algorithms we present in the following three sections (i.e., iteration decomposition, angle decomposition, and pipelined angle decomposition) ensure that these computationally complex stages are parallelized optimally.

Fine-grained decompositions of the DFT beamformer were not attempted since such a solution causes an excess of communication, which is difficult to support on a distributed-memory multicomputer. Such fine-grained decompositions include partitioning each node's FFT and window factor multiplication stages, which would be ill-suited for a loosely coupled system.

3. Iteration decomposition

The first parallel algorithm for in-array DFT beamforming focuses on the partitioning of iterations, where an iteration is defined as one complete beamforming cycle. Iteration decomposition of the DFT beamformer is based on a coarse-grained scheduling algorithm that pipelines the steering factor multiplication, beamform summation, and IFFT/power calculation stages. The window factor multiplication and FFT stages take advantage of the distribution of data across sonar nodes, transforming the data before relaying it to a scheduled processor. This initial data-parallel approach is adapted for each of the other parallel algorithms as well. Intuitively, it is often wise to decompose the program to take advantage of the proximity between available data and processors if that data does not have dependencies. Using this intuition, one might choose to compute the steering factor multiplication stage at each node for local data since data dependencies do not exist until the beamform summation operation. However, the data space that would need to be communicated later for the summation would be many times larger causing significant communication complexity. This characteristic is due to the fact that, for each input signal, the steering factor multiplication stage calculates a phased vector for every steering direction of interest. Therefore, not only is it desirable to parallelize the steering factor multiplication stage to reduce computational complexity, but in so doing we can also reduce memory and communication requirements.

With the iteration-decomposition parallel algorithm, each iteration (from the steering factor multiplication stage through the end of the iteration) is scheduled to a processor in a round-robin fashion. At startup, *node* 1 is scheduled the first iteration, *node* 2 is scheduled the second, etc. The decomposition is split into two computational stages, as shown in Fig. 5, which are referred to as *computation stages* 1 and 2. The first stage is partitioned in the data-parallel fashion previously mentioned while the second stage uses a control-decomposition approach. Therefore, computation stage 1 requires the computational resources from all the nodes, while computation stage 2 finishes the algorithm on a single processor. To increase the efficiency of the algorithm, iterations are pipelined such that new iterations begin during the computation stage 2 for previously started iterations. The pipeline works by interrupting

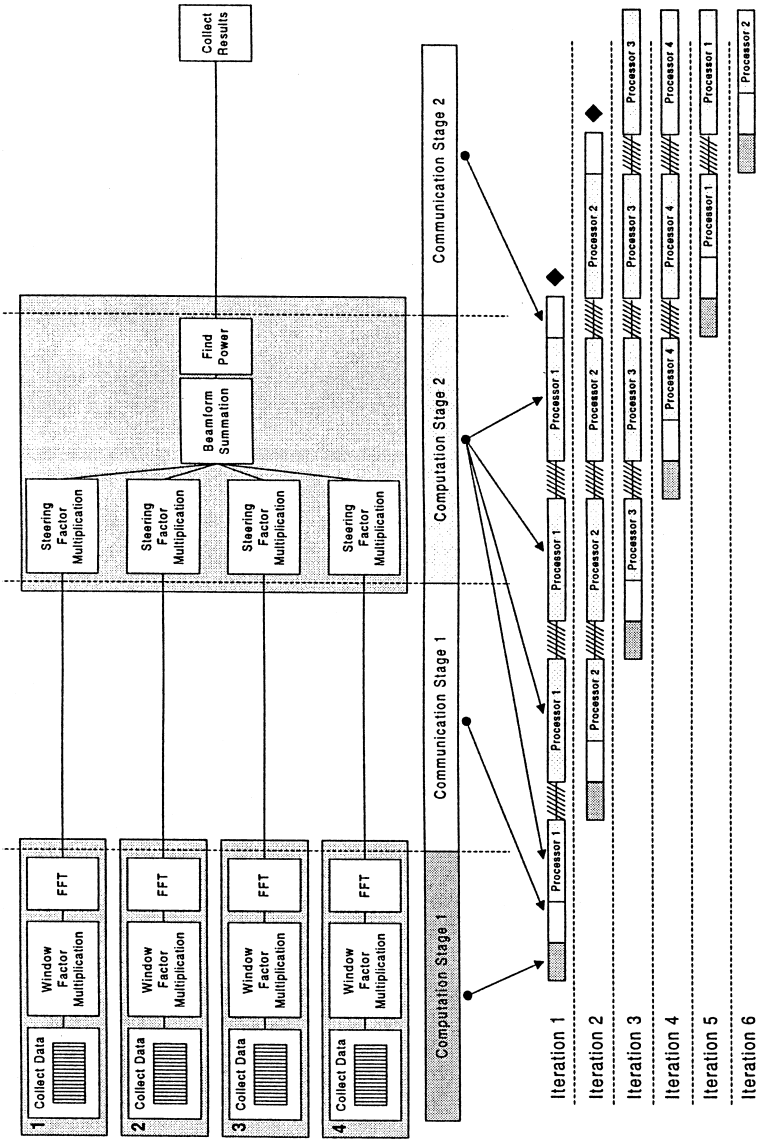


Fig. 5. Iteration decomposition.

iteration's computation stage 2 at well-defined points to begin the computation stage 1 of a new iteration. A 4-node array would require three interruptions to start three new iterations before the computation stage 2 from any one iteration is completed. Thus, the size of the pipeline depends on the number of nodes in the array.

Iteration decomposition normalizes the amount of computation required between interruptions. This trait is a result of the algorithm's linear dependence on number of nodes and is illustrated by the following example. Consider an 8-node array requiring $S = 32$ steering directions and $N = 10$ frequency bins. The steering factor multiplication would require

$$S \times N \times M = 32 \times 10 \times 8 = 2560 \tag{2.7}$$

complex multiplications. However, the operations are pipelined over 8 iterations during which the processors only need to compute 1/8th of the complex multiplies (or 320) per new iteration. If the system was scaled to 16 nodes, then the number of total complex multiplies between interruptions would double. However, the processor would have twice the amount of time to compute the result, maintaining 320 complex multiplies per new iteration over a total of 16 pipelined iterations. Ignoring communication, an array that could support eight nodes could support infinitely many! However, the result latency of any given iteration, which is the time from the original data collection for that iteration until the result is complete, also increases linearly with the size of the array and may conceivably be too long for very large arrays.

The amount of communication also increases linearly with the number of array nodes. More precisely, the complexity of communication stage 1, which contains the communication of the transformed input vectors to the scheduled processor, increases linearly with the number of nodes (M). Communication stage 2, which contains the communication of the results to a designated I/O node, is independent of M. It may be possible to support arbitrarily sized systems by using interconnects such as register-insertion rings, which have the ability to scale well [20]. The communication capability of such interconnects increases as nodes are added. Other less sophisticated networks such as buses are limited with respect to their ability to scale, which increases in array size, since their peak bandwidth is fixed and adding nodes increases contention without increasing aggregate network throughput.

To evaluate the performance of this parallel algorithm, several implementations were coded in C with the message-passing interface (MPI) [13] and executed on a cluster of UltraSPARC workstations connected by 155 Mb/s ATM network via TCP/IP. For each array size, the top of the stacked bars in Fig. 6 represents the average execution time of any given iteration. In Fig. 6(a), the results from computing 91 steering directions are shown, while in Fig. 6(b) the results for 181 steering directions are shown. The times are broken down in stages showing significant computational and communicational operations. As discussed in Section 2, the steering factor multiplication and beamform summation stages dominate the computation times. The time spent in the window factor multiplication/

FFT and power calculation stages is negligible compared to the more significant of the computational stages. The window factor multiplication/FFT stage is so small that it is barely visible in the bottom of each bar graph. There is also significant overhead incurred from communication latencies, which is undoubtedly a result of running the algorithm through the overly robust TCP/IP software layers on top of ATM. We can expect better performance with lighter communication software layers on embedded systems. For instance, with embedded MPI implementations the processes communicate with one another over the network without the need for the internetworking functionality of the TCP/IP protocol stack, and are able to achieve latencies approaching that of the hardware (i.e., the sum of the hardware interface, transmission, and propagation delays). In doing so, communication latency can be reduced from thousands to hundreds of microseconds and less.

Although the result latency for an iteration-decomposition solution is fairly long, the average execution time for any iteration does very well. Iteration decomposition also yields speedup over the sequential program with a parallel efficiency of about 68%, as shown in Fig. 7. Because the problem size increases with the number of nodes, these speedup numbers reflect scaled speedup. Furthermore, the comparisons are made to a purely sequential algorithm that ignores communication latency, as previously discussed.

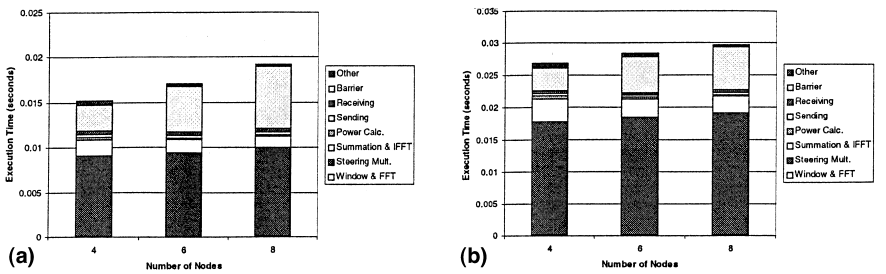


Fig. 6. Average execution times for the iteration-decomposition method (averaged over 1000 iterations). (a) 91 steering direction, (b) 181 steering direction.

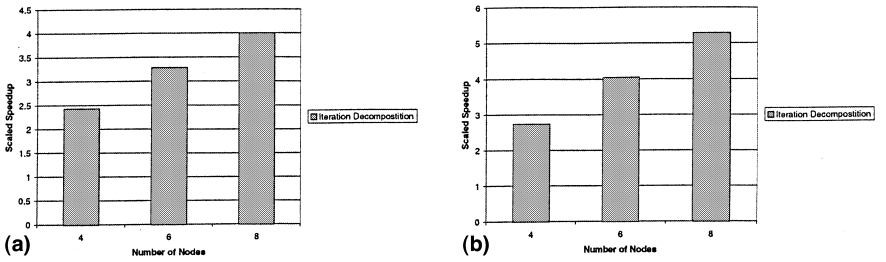


Fig. 7. Scaled speedup for the iteration-decomposition method. (a) 91 steering direction, (b) 181 steering direction.

4. Angle decomposition

The second parallel algorithm focuses on partitioning steering angle solutions to each of the nodes, a form of domain decomposition. Angle decomposition of the parallel DFT beamforming algorithm is based on a medium-grained algorithm that partitions the five operations discussed in Section 2 across all nodes. The window factor multiplication and FFT stages (or computation stage 1) again take advantage of the distribution of data across sonar nodes and operate in exactly the same manner as in iteration decomposition. On the other hand, computation stage 2 differs significantly from the method used in the prior decomposition. In angle decomposition, the entire algorithm operates in a data-parallel fashion. Instead of a single node computing the beamform solution for all angles, the nodes divide the S steering directions among the processors and independently beamform in those directions. The number of steering angles each node computes is thus S/M. Using this decomposition, the workload for a single iteration is distributed evenly to all nodes.

Fig. 8 shows a block diagram illustrating this method. With respect to communication stage 1, angle decomposition is immediately distinguishable from iteration decomposition. As shown in the figure, the communication in this stage is an all-to-all communication, which is this algorithm's greatest deficiency. The communication does not scale linearly but rather quadratically with the number of nodes, resulting in a more complex $O(M^2)$ communication. However, with some broadcast-capable networks, this complex communication can be reduced to $O(M)$. For instance, slotted ring networks can be employed in a sonar array system. With a slotted ring, each node is guaranteed its fair share of the aggregate bandwidth, and for each node to communicate with all other nodes it merely transmits a single message with a broadcast header onto the network. Each of the other nodes on the ring copies and uses the same data from the message and after full circulation around the ring the message is removed by the sender. In this fashion, the complexity of the all-to-all communication is reduced from $O(M^2)$ to $O(M)$ since each node communicates but once. As in the previous algorithm, the total amount of communication in communication stage 2 is independent of array size, although angle-decomposition partitions the output data into M smaller segments.

One advantage of this algorithm over iteration decomposition is the low result latency of any individual beamform solution. Recall that with iteration decomposition, each solution's latency is dependent on the size of the array. That is, larger arrays will cause more interruptions for starting new iterations, thus increasing the length of time required to finish computations for any given iteration. Angle decomposition ensures the lowest latency to produce a final result of any decomposition. Using much the same reasoning as for iteration decomposition, the latency of the beamform solution is computationally independent of the array size, a result that is due to the algorithm's linear dependence on M. This linear dependence implies that the latency of any single solution of the algorithm will be the same for eight nodes as it is for infinitely many, again ignoring communicational requirements. If the interconnect employed was fully connected or could scale as the square of M, then the

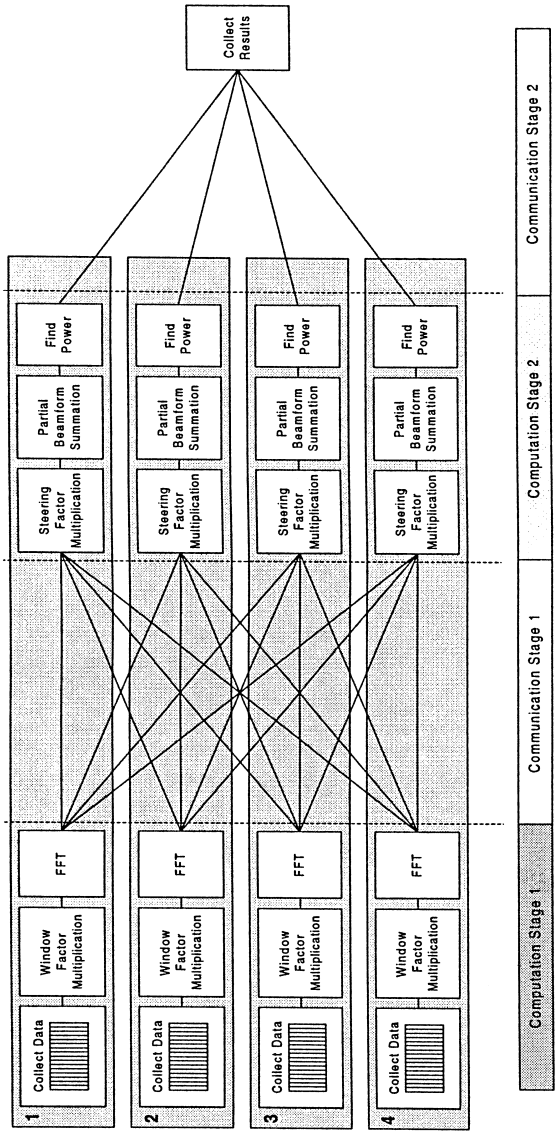


Fig. 8. Angle decomposition.

network would also support linear scalability [9]. However, for many applications such a robust network is unlikely to be cost effective.

Another advantage of the angle-decomposition parallel algorithm is the efficient use of memory. Since each node performs a range of steering angles, memory required for the steering factor multiplication may be distributed across all nodes. The iteration-decomposition algorithm requires either recomputation of the large steering matrix with each new iteration or copies of the matrix in every node.

For the domain decomposition used in this algorithm, the degree-of-parallelism (DOP) achieved is dependent on the number of steering directions, with high-resolution beamformers expected to glean the best speedup results. The number of frequency bins also increases the DOP but, similar to the number of nodes, has the unfortunate effect of increasing the communication requirements quadratically.

Fig. 9 charts the average execution times for computing 91 steering directions and 181 steering directions. The times are again broken down in stages to show significant computational and communicational operations. The steering factor multiplication and beamform summation stages dominate the computation times while the FFT and window and power calculation stages are negligible. The communication in

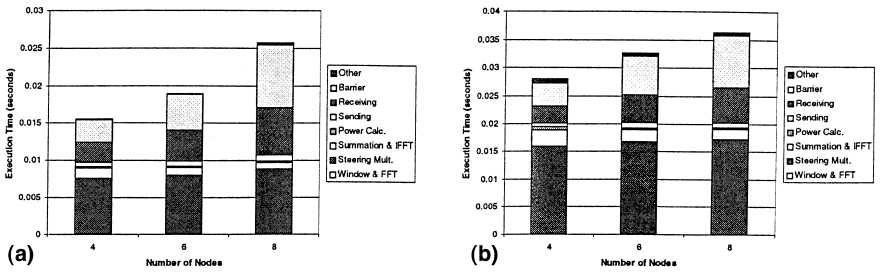


Fig. 9. Average execution times for the angle-decomposition method (averaged over 1000 iterations). (a) 91 steering direction, (b) 181 steering direction.

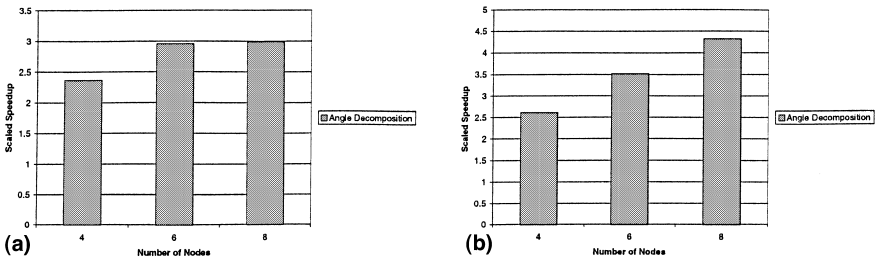


Fig. 10. Scaled speedup of the angle-decomposition method. (a) 91 steering direction, (b) 181 steering direction.

the angle-decomposition algorithm represents more than half of the total execution time. Again, these large communication latencies may be exaggerated in comparison to the actual hardware latencies since an in-array processing system will have a lighter communication stack than TCP/IP.

As shown in Fig. 10, the scaled speedup of the angle decomposition is relatively poor, yielding a parallel efficiency of less than 50% in most cases. The large communication latencies were detrimental to the overall performance of the algorithm.

5. Pipelined angle decomposition

The third parallel algorithm focuses on partitioning steering angle solutions to each of the nodes (i.e., data parallelism), but it also employs pipelining (i.e., control parallelism) to improve efficiency. Pipelined angle decomposition overlaps the communication and computational stages of angle decomposition at the expense of higher result latency for any single beamform iteration. Therefore, pipelined angle decomposition is a compromise between the iteration- and angle-decomposition algorithms. Pipelined angle decomposition decreases result latency from that in iteration decomposition and achieves better speedup than angle decomposition.

The pipelining, shown in Fig. 11, is achieved by overlapping communication stage 1 with computation stage 2 from the preceding iteration and computation stage 1 from the succeeding iteration. The result collection in communication stage 2 is overlapped in a similar fashion. After iteration's communication stage 1 is initiated, the algorithm picks up the previous iteration at computation stage 2. At the end of this computational stage, the result collection, communication stage 2, for that iteration is begun. Finally, before completing communication stage 1, the algorithm begins computation stage 1 of a new iteration. Thus, an effective overlapping of communication and computation stages is achieved.

Pipelined angle decomposition is not as efficient as iteration decomposition but constrains the length of the pipeline to three iterations, as opposed to iteration decomposition whose latency grows as a function of M. Therefore, ignoring the increasing communication latencies, the result latency to produce a single beamform solution in pipelined angle decomposition remains independent of array size, yet is longer than the latency in pure angle decomposition. The DOP of the algorithm remains identical to that of angle decomposition, but average speedup is improved.

In Fig. 12, the execution times for computing 91 steering directions and 181 steering directions are shown. As expected, the figures show that the pipelining has improved the average execution time from that of angle decomposition. The steering factor multiplication and beamform summation still account for the majority of computational load while the execution times of FFT, window factor multiplication and power calculation stages are still insignificant. Communication latencies of

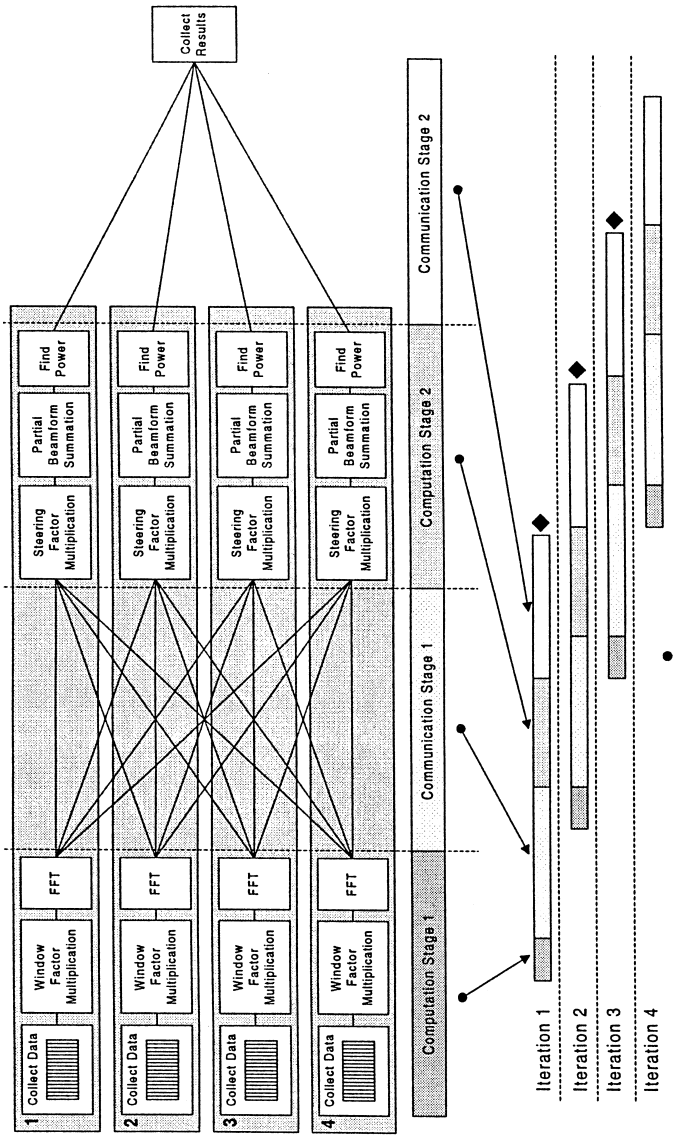


Fig. 11. Pipelined angle decomposition.

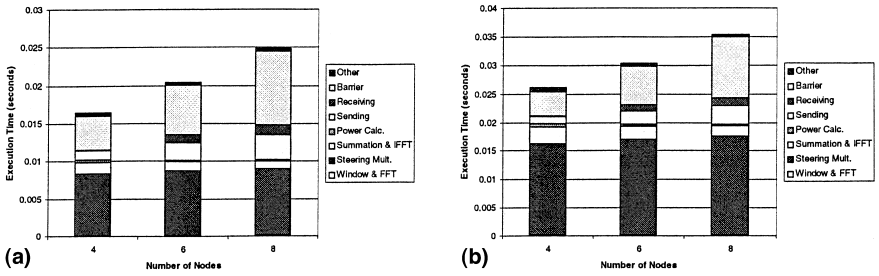


Fig. 12. Average execution times for pipelined angle decomposition (averaged over 1000 iterations). (a) 91 steering direction, (b) 181 steering direction.

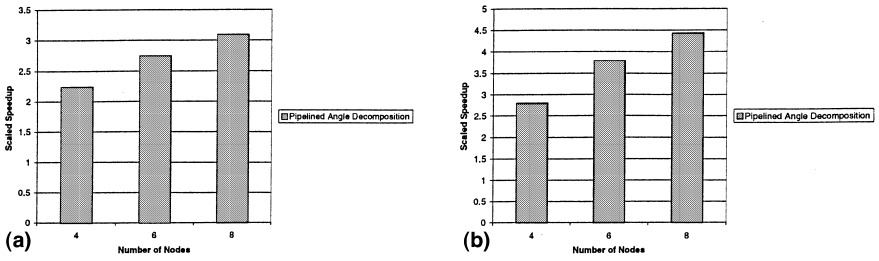


Fig. 13. Scaled speedup for the pipelined angle decomposition. (a) 91 steering direction, (b) 181 steering direction.

pipelined angle decomposition are minimized, which is clearly shown by the time spent receiving with respect to angle decomposition.

The scaled speedup for pipelined angle decomposition, shown in Fig. 13, has improved marginally over angle decomposition for large array sizes by increasing parallel efficiencies to over 50%. Small arrays take advantage of the overlap in communication and computation, increasing parallel efficiency to almost 70%.

6. Comparative analysis

In this section, the advantages and disadvantages of each of the three algorithms are discussed, and comparisons are made between their performances. Fig. 14 shows the average execution time for each of the parallel beamformers. The iteration-decomposition algorithm shows the best performance, whereas the angle-decomposition algorithm generally performs the worst. Of course, the trade-off between these two algorithms is the large result latency incurred for iteration decomposition. The pipelined angle decomposition, as predicted in Section 5, compromises the angle decomposition's short result latency for a somewhat better average execution time. Large array sizes may cause the communication in the two angle-decomposition algorithms to grow quickly, especially if the network

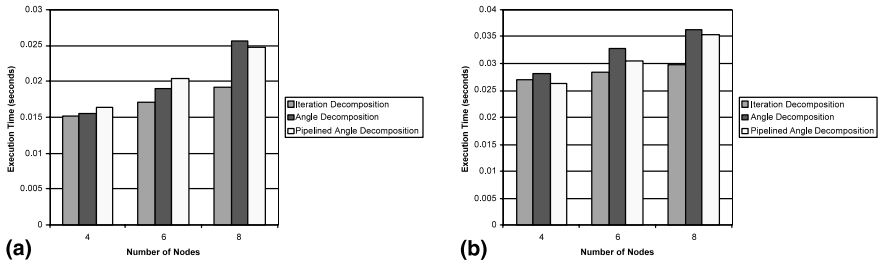


Fig. 14. Average execution times for the three parallel algorithms (averaged over 1000 iterations). (a) 91 steering direction, (b) 181 steering direction.

cannot support broadcast traffic. A study of this effect is discussed in more detail below.

The scaled speedup plots in Fig. 15 indicate that iteration decomposition delivers the best performance of the three algorithms with pipelined angle decomposition performing almost as well in some cases. Although angle decomposition is conceptually less efficient than its pipelined cousin, the added complexity of setting up the pipelined angle decomposition led to more efficient execution by angle decomposition for small arrays with lower steering resolution. Angle decomposition certainly performs less favorably for larger arrays and higher-resolution systems.

To provide a sensitivity study of array parameters on an in-array processing system, each of the parallel beamformers was also executed on the integrated simulation environment (ISE), a rapid virtual prototyping tool developed at the University of Florida [6]. ISE has the ability to run real applications written in MPI over simulated systems built in the block oriented network simulator (BONeS), a commercial product of Cadence Design Systems. A number of interconnection schemes have been developed specifically in ISE for prototyping a distributed parallel sonar array. These interconnects include a register insertion ring, a bidirectional register insertion array, and a slotted ring, each of which support linear scalability with increasing numbers of nodes.

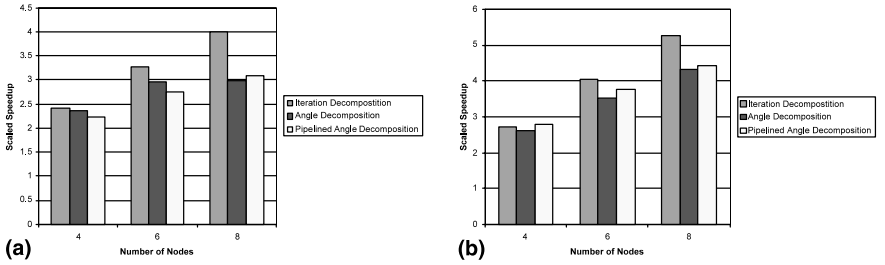


Fig. 15. Scaled speedups for the three parallel algorithms. (a) 91 steering direction, (b) 181 steering direction.

ISE allows researchers to experiment with systems that may be too impractical or expensive to prototype in the traditional sense. The following experiments show the sensitivity and scaling ability of the three parallel beamforming algorithms over a larger range of parameters. From this data, a more accurate account of how the decompositions will perform on an actual in-array processing system is engaged.

The first sensitivity study involves the variation of parameters for network speed and processor speed in an 8-node array with a bidirectional register-insertion network using a minimal protocol stack. To study the effect of the speed of the eight processors in the virtual prototype, variations from 25% to 100% of the performance of an UltraSPARC processor running at 170 MHz are used. In addition, variations in network speed range from 2.5 to 10 Mb/s. Both ranges are representative of the levels of performance that can be expected with low-power components for computation and communication. Each of the three parallel beamformers was executed on virtual prototypes with these permutations, and execution times are shown in Fig. 16. As can be seen, the iteration and angle-decomposition methods provide similar performance results as the speed of the system is varied. The processor speed has the most significant effect on the execution time of these beamformers. As network speed is decreased, the performance of the algorithms is also decreased, though to a lesser extent. Graphically, this trend appears as a gradual slope in the network-

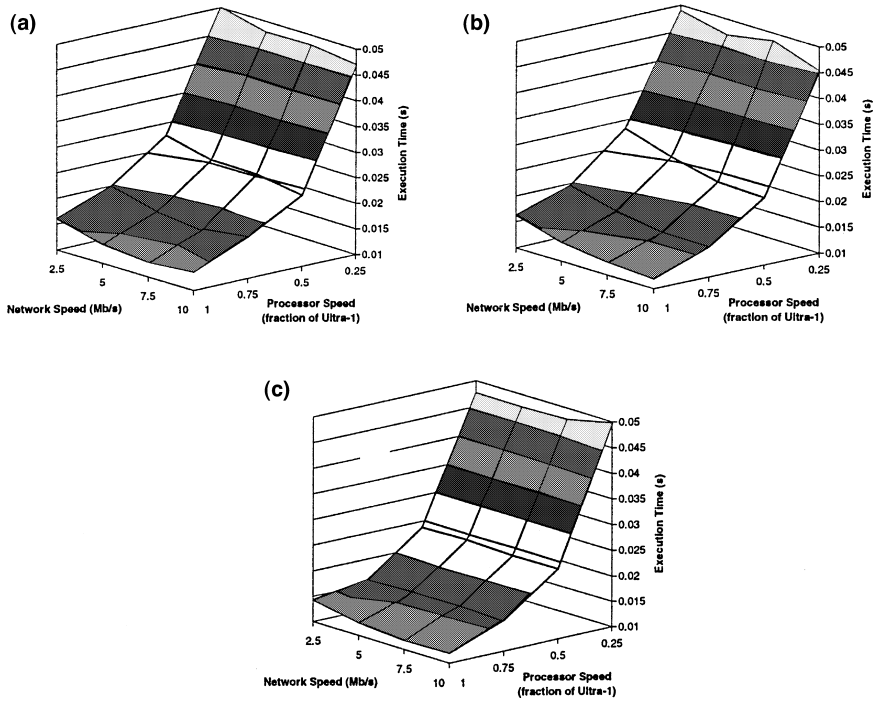


Fig. 16. Contour plots for each algorithm ranged over network and processor speed. (a) Iteration decomposition, (b) angle decomposition, (c) pipelined angle decomposition.

speed dimension. However, pipelined angle decomposition shows little performance change as network speed is varied, indicating the additional ability of this method to overlap computation with communication. However, this result is likely to change for very large arrays, in which communication latencies may be more dominant than computational latencies. Dependency on network speed for this 8-node system only appears when the processor speed is so fast and the network speed so slow as to cause the communication to last longer than the computations. As with the other algorithms, processor speed remains the dominant constraint for pipelined angle decomposition.

The second sensitivity study, shown in Fig. 17, demonstrates the scalability of the algorithms as the number of nodes is changed. The execution time increases linearly as the number of nodes is increased; however, the slope is gradual for all three algorithms. This trend bodes well for the angle-decomposition methods, which might be expected to incur large communication latencies for large arrays. From 2 to 32 nodes, the execution time per iteration increases approximately 30% for all decompositions. Since the problem size and the processing capacity both grow linearly with the number of nodes for a fixed number of steering directions and frequency bins, as the problem size doubles the increase in execution time would ideally be 0%. However, due to the communication and the synchronization required between the processors in the nodes of the array, the results indicate a modest increase in execution time of approximately 12% as the problem size doubles.

7. Conclusions and future research

The continuing development of beamforming algorithms is creating the need for integrated solutions that leverage algorithmic optimizations with parallel embedded system architectures. These architectures must support a large number of nodes, steer with increased fidelity, and offer better fault tolerance in the event of node failures, all of which place increasing strain on memory requirements, processor speed, network efficiency, and software intelligence. Parallel system architectures hold the potential to eliminate single points of failure and improve hardware fault

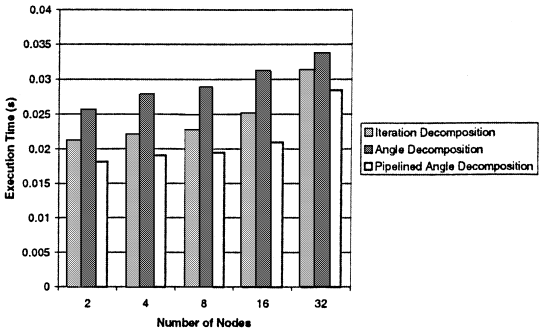


Fig. 17. Execution time for each algorithm vs number of nodes.

masking and tolerance, while parallel algorithms for these architectures can support large arrays due to linear dependence of problem size on the number of nodes. The goal of this research was to provide general solutions for in-array parallel beamforming, first for conventional beamformers, but ultimately extensible to split-aperture, adaptive, and matched-field techniques.

The configuration of a sonar array maps particularly well to loosely coupled multicomputer architectures, thus the parallel solutions were constrained to coarse-grained and medium-grained decompositions. Although fine-grained solutions are not impossible, they are certainly inefficient on these systems. Using the sequential DFT beamformer as a baseline, new parallel algorithms were developed using both coarse-grained iteration and medium-grained angle decomposition. In addition, by combining attributes of both iteration and angle decomposition, a hybrid form of parallel algorithm was developed.

The iteration-decomposition algorithm was found to exhibit the best efficiency and the best scalability at the cost of a large result latency. Of the algorithms presented, it is also the least complex to implement and is easily adaptable in the event of hardware faults. The angle-decomposition method shows the lowest result latency and is memory-efficient since the steering factor multiplication matrix may be distributed across the entire array. However, it also exhibits the worst parallel efficiency and may be difficult to restructure in the event of node failures. The pipelined angle decomposition improves the efficiency of angle decomposition at the cost of longer result latency and maintains the memory efficiency of the angle-decomposition algorithm. However, it also suffers from difficulty in supporting fault-tolerant procedures. Each of the parallel beamforming algorithms presented is scalable to different degrees for large arrays, as was shown by sensitivity analyses with the aid of a rapid virtual prototyping tool.

Future directions from this research will focus on leveraging the contributions from these new parallel algorithms for in-array processing to support additional beamformers with increasing sophistication in their acoustic and signal processing attributes. Currently, extensions under development include parallel algorithms for split-aperture beamforming and MVDR adaptive beamforming. Furthermore, research on the implementation of fault-tolerance mechanisms to support the three parallel algorithms is underway.

Acknowledgements

The support provided by the Office of Naval Research on grant N00014-98-1-0188 is acknowledged and appreciated.

References

[1] S. Banerjee, P.M. Chau, Implementation of adaptive beamforming algorithms on parallel processing DSP networks, Proceedings of SPIE – The International Society for Optical Engineering 1770 (1992) 86–97.

[2] P.M. Clarkson, Optimal and Adaptive Signal Processing, CRC Press, Ann Arbor, 1993.
[3] R.F. Dwyer, Automated real-time active sonar decision making by exploiting massive parallelism, IEEE Oceans Conference Record 3 (1996) 1193–1196.
[4] M.J. Flynn, Some computer organizations and their effectiveness, IEEE Transactions on Computers 21 (9) (1972) 948–960.
[5] O.L. Frost III, An algorithm for linearly constrained adaptive array processing, Proceedings of IEEE 60 (1972) 926–935.
[6] A. George, R. Fogarty, J. Markwell, M. Miars, An integrated simulation environment for parallel and distributed system prototyping, Simulation 72 (5) (1999) 283–294.
[7] S. Haykin, Array Signal Processing, Prentice-Hall, Englewood Cliffs, NJ, 1985.
[8] K.M. Houston, A fast beamforming algorithm, IEEE Oceans Conference Record 1 (1994) 211–216.
[9] K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw-Hill, New York, 1993.
[10] W.C. Knight, R.G. Pridham, S.M. Kay, Digital signal processing for sonar, Proceedings of IEEE 69 (11) (1981) 1451–1506.
[11] F. Machell, Algorithms for broadband processing and display, Applied Research Laboratories Technical Letter No. 90-8 (ARL-TL-EV-90-8), Applied Research Laboratories, The University of Texas at Austin, 1990.
[12] D. McMahon, A. Bolton, Signal-to-noise ratio enhancement of cyclic summation, Proceedings of the International Symposium on Signal Processing and its Applications (ISSPA) 2 (1996) 710–713.
[13] Message-Passing Interface Forum, MPI: A Message-Passing Interface Standard, University of Tennessee, Knoxville, TN, May 1994.
[14] D. Morgan, Practical DSP Modeling Techniques and Programming in C, Wiley, New York, 1994.
[15] R.A. Mucci, A comparison of efficient beamforming algorithms, IEEE Transactions on Acoustics Speech, and Signal Processing ASSP-32 (3) (1984) 548–558.
[16] R. Nielsen, Sonar Signal Processing, Artech House, Boston, 1991.
[17] J. Salinas, W.R. Bernecky, Real-time sonar beamforming on a MasPar architecture, in: Proceedings of the IEEE Symposium on Parallel and Distributed Processing, 1996, pp. 226–229.
[18] W.W. Smith, J.M. Smith, Handbook of Real-Time Fast Fourier Transforms: Algorithms to Product Testing, IEEE Press, New York, 1995.
[19] W. Robertson, W.J. Phillips, A system of systolic modules for the MUSIC algorithm, IEEE Transactions on Signal Processing 41 (2) (1991) 2524–2534.
[20] W. Stallings, Data and Computer Communications, Macmillan, New York, 1994.
[21] C.R. Ward, P.J. Hargrave, J.G. McWhirter, A novel algorithm and architecture for adaptive digital beamforming, IEEE Transactions on Antennas and Propagation AP-34 (1986) 338–346.
[22] G.P. Zvara, Real time time–frequency active sonar processing: a SIMD approach, IEEE Journal of Oceanic Engineering 18 (1993) 520–528.